Net-Centric Implementation

Part 1: Overview

Part 2: ASD(NII) Checklist Guidance

Part 3: Migration Guidance

Part 4: Node Design Guidance

Part 5: Developers Guidance

Part 6: Acquisition Guidance

V 1.2

20 December 2005

This document is a NESI product.
NESI (Net-Centric Enterprise Solutions for Interoperability) is a collaborative activity between the USN PEO for C4I and Space and the USAF Electronic Systems Center.

Table of Contents

1 NESI implementation

1.1 References

(a) DoD Directive 5000.1, *The Defense Acquisition System*, 24 November 2003.

(b) DoD Instruction 5000.2, *Operation of the Defense Acquisition System*, 12 May 2003.

(c) DoD Directive 8100.1, *Global Information Grid (GIG) Overarching Policy*, 21 November 2003.

(d) DoD Directive 4630.5, *Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS)*, 05 May 2004.

(e) DoD Instruction 4630.8, *Procedures for Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS)*, 30 June 2004.

(f) DoD Directive 5101.7, *DoD Executive Agent for Information Technology Standards*, 21 May 2004.

(g) *DoD Global Information Grid (GIG) Architecture, Version 2.0*, August 2003.

(h) *DoD Joint Technical Architecture, Version 6.0*, 3 October 2003.

(i) *DoD Net-Centric Data Strategy*, DoD Chief Information Officer, 9 May 2003.

(j) CJCSI 3170.01D, *Joint Capabilities Integration and Development System*, 12 March 2004.

(k) CJCSM 3170.01A, *Operation of the Joint Capabilities Integration and Development System*, 12 March 2004.

(l) CJCSI 6212.01C, *Interoperability and Supportability of Information Technology and National Security Systems*, 20 November 2003.

(m) *Net-Centric Operations and Warfare Reference Model (NCOW RM) V1.0*, September 2003.

(n) *Net-Centric Checklist, V2.1.3*, Office of the Assistant Secretary of Defense for Networks and Information Integration/Department of Defense Chief Information Officer, 12 May 2004.

(o) *A Modular Open Systems Approach (MOSA) to Acquisition, Version 2.0*, September 2004.

(p) DoD IT Standards Registry (DISR), http://disronline.disa.mil.

(q) *Net-centric Attributes List*, Office of the Assistant Secretary of Defense for Networks and Information Integration/Department of Defense Chief Information Officer, June 2004.

1.2 Overview

Net-centric Enterprise Solutions for Interoperability (NESI) **is a joint effort between the U.S. Navy's Program Executive Office for C4I & Space and the U.S. Air Force's Electronic Systems Center. It provides implementation guidance which facilitates the design, development, maintenance, evolution, and use of information systems for the Net-Centric Operations and Warfare (NCOW) environment. NESI has also been provided to other Department of Defense (DoD) services and agencies for potential adoption.**

The NESI Implementation guidance applies to all phases of the acquisition process as defined in references (a) and (b). NESI comprises six parts, each focusing on a specific area of guidance. *NESI Part 1: Net-centric Overview* describes each part in detail.

NESI provides guidance, best practices, and examples for developing Net-Centric software. It is aligned with the design principles of reference (o). NESI is not a replacement for references (m), (n), or (p).

The overall goal is to provide common, cross-service guidance in basic terms for the program managers and developers of net-centric solutions. The objective is not to replace or repeat existing direction, but to help translate into concrete actions the plethora of mandated and sometimes contradictory guidance on the topic of net-centric compliance and standards.

NESI subsumes two now obsolete references; in particular, the Air Force *C2 Enterprise Technical Reference Architecture (C2ERA)*[1] and the Navy *Reusable Applications Integration and Development Standards (RAPIDS)*.[2] Initial authority for NESI is per the Memorandum of Agreement between Space and Naval Warfare Systems Command (SPAWAR), Navy PEO C4I & Space and the United States Air Force Electronic Systems Center, dated 22 December 2003, Subject: Cooperation Agreement for Net-Centric Solutions for Interoperability (NESI).

In addition to references (a) through (q), Navy PEO C4I & Space has mandated a software maintenance policy[3] for its programs that requires the use of *NESI Part 3: Net-Centric Migration Guidance*.

NESI is intended to help programs comply with the DoD net-centric directives, instructions, and other guidance documentation (listed as references (a) through (q) above). This guidance will continue to evolve as direction and our understanding of the requirements of net-centricity evolve. NESI will be updated to reflect changes to the guiding documents and new regulations.

1.3 Releasability statement

This document has been cleared for public release by competent authority in accordance with DoD Directive 5230.9 and is granted Distribution Statement A: Approved for public release; distribution is unlimited. You may obtain electronic copies at https://nesipublic.spawar.navy.mil.

---

[1] Air Force C2 Enterprise Technical Reference Architecture, v3.0-14, 1 December 2003.

[2] RAPIDS Reusable Application Integration and Development Standards, Navy PEO C4I & Space, December 2003 (DRAFT V1.5)

[3] Software Maintenance Policy, Department of the Navy, PEO C4I & Space, 14 June 2004.

1.4 Vendor neutrality

The NESI documentation sometimes refers to specific vendors and their products in the context of examples and lists. However, NESI is vendor-neutral. Mentioning a vendor or product is not intended as an endorsement, nor is a lack of mention intended as a lack of endorsement.

Code examples typically use open-source products, since NESI is built on the open-source philosophy. Since NESI accepts contributions from multiple sources, the examples also tend to reflect whatever tools the contributor was using or knew best. However, the products described are not necessarily the best choice for every circumstance. You are encouraged to analyze your specific project requirements and choose your tools accordingly. There is no need to obtain, or ask your contractors to obtain, the open-source tools that appear as examples in this guide. Similarly, any lists of products or vendors are intended only as references or starting points, and not as a list of recommended or mandated options.

1.5 Disclaimer

Every effort has been made to make this documentation as complete and accurate as possible. It is expected that the documentation will be updated frequently, and will not always immediately reflect the latest technology or guidance.

1.6 Contributions and comments

NESI is an open-source project that will involve the entire development community. Anyone is welcome to contribute comments, corrections, or relevant knowledge to the guides. To submit comments, corrections, or contributions go to the NESI public site at http://nesipublic.spawar.navy.mil and click on the Change Request tab, or sent an email to nesi@hanscom.af.mil or nesi@spawar.navy.mil.

1.7 Open-source site

PEO C4I & Space is in the process of establishing an open-source site to support community involvement. Use this site for collaborative software development across distributed teams. Check the NESI public site for updates on when the collaborative development site will be available.

2 Introduction

Moving to a net-centric environment is a high priority of DoD leadership. NESI is taking a lead role in executing that vision. However, there are few or no additional dollars available for net-centricity. This requires us to use current resources more effectively. To successfully transition to a net-centric environment, programs need guidance that provides clear objectives and suggests "inch stone" improvements that can occur in conjunction with routine maintenance activity.

To move applications towards this goal, NESI advocates an incremental migration strategy. Programs and contracts should use existing maintenance dollars to migrate applications to a system capable of net-centric warfare (NCW) while meeting current maintenance obligations. This approach leverages the DoD's investment in deployed systems and training.

This document provides the following guidance:

- Section 3, *Migrating to a net-centric warfare environment*: Defines incremental migration strategies tailored to the type of application.
- Section 4, *Selecting a migration level*: Provides tools for assessing the risk and scope of migration, identifies three levels of upgrade, then discusses how to implement upgrades in each tier.
- Section 5, *Migrating COE systems and applications*: Recommends approaches to migrating COE-based systems to a net-centric infrastructure.
- Section 6, *Mapping maintenance actions to enterprise technology objectives*: Maps the maintenance actions described in section 4 to the enterprise technology objectives described in *NESI Part 1: Net-Centric Overview*, such as capability on demand.

2.1 Audience

The intended audience for this document includes:

- Program managers
- Deputy program managers
- Contracting officers
- Chief engineers
- Contractor personnel

3 Migrating to a net-centric warfare environment

The technical migration strategy outlined in this document is built around a planned migration of functionality out of the current stovepipe systems and COE segments into a set of componentized applications, N-tier, layered systems, separate node-based infrastructures, and new services. This migration strategy is based on fiscal limitations and the need to continue supporting current applications.[4]

Application functionality will be duplicated or wrapped, not necessarily removed, from current systems until all clients are using services instead of legacy stovepipe systems. Not all upgrades need to occur at the same time. Developers should identify and refactor application logic during planned maintenance activities. When application code is updated in a specific area, they should take the opportunity to add the appropriate net-centric environment upgrades.

This concurrent strategy entails insulating the structure of each system/application so they can proceed independently of other enterprise systems and applications. *NESI Part 5: Net-Centric Developers Guide* discusses insulation techniques such as multi-tier architecture, connectors, wrappers, adaptors, facades, proxies, bridges, and abstract interface classes.

In many cases, incremental migration is more efficient and carries less risk than a direct cutover. The benefits include:

- Staying within the current acquisition frameworks
- Leveraging common development opportunities
- Providing reasonable incentive for participation
- Offering short-term, tangible results that do not disrupt ongoing software development efforts
- Reducing cost as opposed to an expensive stop and re-engineer strategy

The main risk of this approach is that people may try to apply a one-size-fits-all strategy to all programs. This section discusses how to mitigate this risk by tailoring the migration to different types of programs.[5] To that end, we define “migration levels” that identify different levels of adaptation and map them to enterprise technology objectives.

3.1 ASD (NII)/DoD CIO categories

Reference (n) assigns programs to net-centric categories. The following sections identify each category and list relevant documentation.

The effort required to implement net-centricity in an application varies based on the target application. The spectrum ranges from already web-enabled, multi-tiered, service-based applications to single-tiered, proprietary, closed stovepipes. In some cases, developers may wrap

[4] This initial policy document focuses on software-related systems. More detailed network and transport level guidance will follow.

[5] Real-time and closed-loop systems will implement a separate set of guidance (e.g., Open Architecture) that may not share all of the net-centric attributes.

the entire application into one or several high-level coarse-grained interfaces. Then, the system components can migrate during subsequent iterations.

3.1.1 Non-compliant (Retire)

Programs that do not exhibit net-centric capabilities and are not essential for continued operations or business processes will be guided towards termination.

3.1.2 Legacy being sustained (Retain)

Current programs that do not yet exhibit net-centric capabilities and are not planned for transformation, but are essential for current operations. If minimal cost growth is obtainable, programs in sustainment mode should attempt to meet the minimum criteria. ***This document applies to some programs in this category, on a case-by-case basis.***

3.1.3 Legacy being transformed (Refresh)

Current programs that have an established plan to comply with net-centric capabilities and DoD domain requirements. ***This document specifically addresses programs in this category.***

The matrix in section 6, *Mapping maintenance actions to enterprise technology objectives,* explains how to make net-centric enhancements that are of the same scope as typical maintenance or fix actions. The objective is to stretch existing maintenance dollars to build net-centric capabilities as well. If more money is available, programs can achieve higher objectives.

3.1.4 New start / In development

Programs that are born net-centric. They meet NCW requirements and are fully compliant with DoD net-centric models.

3.2 NESI migration levels

The table below illustrates the relationship of the ASD (NII)/DoD CIO categories to the NESI technical migration level.

Table 1 – Correspondence between Checklist and NESI Levels

ASD (NII)/DoD CIO Checklist Categories	NESI Migration Levels
Non-compliant (Retire)	Level 0 (As-Is) – Point to point legacy interfaces
Legacy being sustained (Retain)	Level 1 – Migration to N-tier structure Level 2 – Access to legacy data and applications
Legacy being transformed (Refresh)	Level 3 – Legacy applications transformed
New start/In development	Level 4 – Fully integrated applications and databases

The following figure depicts these levels of migration. This document discusses the technical activities involved in migrating to Levels 1, 2, and 3. *NESI Part 2: Net-centric ASD (NII) Checklist Guidance* presents the criteria for Level 4.

Provide Flexibility through Multiple Levels of Migration

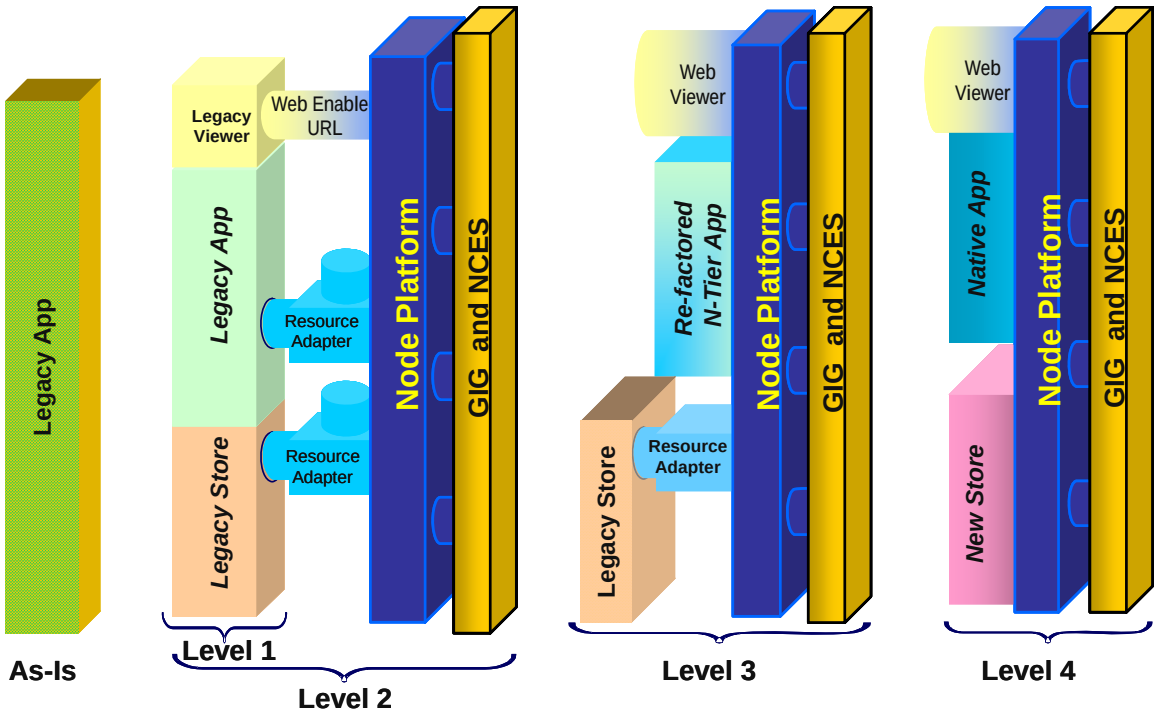


Figure 1 – Levels of Migration

4 Selecting a migration level

This section identifies several application migration levels. Each level lists technical net-centric upgrade actions for each of the enterprise technology objectives listed in *NESI Part 1: Net-Centric Overview*. These upgrades enable programs to implement the technical and application attributes they need to become net-centric and qualify for ASD (NII)/DoD CIO's "Refresh" category.

As one finds when buying a car, packaged options are more economical than random selections of options. We believe that this will also be true for systems migrating to net-centricity. The levels discussed here organize upgrades into logical groups and consider effort, complexity, and cost in a consistent manner.

NESI identifies four major levels of upgrades that range from a minimal upgrade to a full net-centric integration. Program managers select the appropriate level based on risk and resource costs. See section 4.1, *Assessing risk,* and section 4.2, Assessing scope, before identifying a migration level.

- **Level 1, Minimum upgrade**: Factor the application into tiers, modularize application code into components, and create public Application Programming Interfaces (APIs).
- **Level 2, Mid-level upgrade**: Using adapters, connect the application through its public APIs to the Node Platform Infrastructure (NPI).
- **Level 3, Net-centric upgrade**: Migrate application code to the Node Platform Infrastructure and prepare the application for the net-centric environment (namespaces, metadata, XML).
- **Level 4, Full net-centric integration:** Enable legacy applications to function more fully in the net-centric environment. These enhancements are typically more expensive and time-consuming.

The levels are not compliance levels, and programs will not be judged against them. Moreover, program managers are free to select upgrades from multiple levels to suit the needs of their application and the availability of resources.

It is essential to select upgrades that support structural insulation. This approach allows each modification to proceed independently. Carefully consider the best approach to applications that cannot be modified during other maintenance activities.

4.1 Assessing risk

The NESI technical approach is based on migrating applications to an N-tier architecture. The following list describes some risks associated with net-centric upgrades. Consider these when choosing your upgrade level. Applications that migrate from an environment with a single or limited number of users, or that migrate into a cross-domain security environment, require additional evaluation.

4.1.1 Client and presentation tiers

These upgrades carry the lowest risk and the highest return, because they make the application more net-centric without affecting the operation of the application.

Depending on the legacy application architecture, it may be more appropriate to merge the client and presentation tiers. This guidance applies in both cases.

4.1.2 Middle tier

These upgrades entail higher risk for applications that are not component-based or not structured with well-defined interfaces. Mitigate the risk by targeting smaller areas of the middle tier rather than the full application.

4.1.3 Data tier

These upgrades also entail higher risk. The risk is especially high for applications that lack well-defined interfaces, are not insulated from the database or data stream, or store all the business logic in the database. Mitigate the risk by using a combination of approaches such as switching to ODBC/JDBC, removing detailed business logic and algorithms from the data tier and targeting small areas rather than reworking the entire application at once.

There are multiple technologies for exposing data to the enterprise. Providing a data-tier-service API involves significant costs and risks, which you can mitigate with careful analysis and design. Make sure to address data modeling, API construction, data element granularity, and XML format conversion. For example, share information between nodes via middle-tier services and not via SQL over the network.

As an example, for database applications, do not start by pulling all the business logic out of the database. Instead, leave the stored procedures in place and migrate to an open-standards abstraction layer such as ODBC/JDBC. In subsequent iterations, separate detailed business logic and algorithms from the database. DBMS stored procedures, triggers and constraint checks are the optimal approach for inserting data, manipulating data, cascading deletions, enforcing constraints and referential integrity.

For data stream applications, try implementing an open-standards wrapper abstraction layer to cover a subset of protocols. Extend the abstraction layer in subsequent iterations.

4.1.4 Multi-user applications

These upgrades entail significant risk. Scaling a single-user application to be net-accessible and consequently multi-user is quite complex and requires careful planning. Some of the issues that need to be addressed are: concurrency, locking, priority, transactions, state transitions failover, security and logging.

4.1.5 Cross-domain security (CDS)

Adopting CDS is often the highest risk area, since CDS designs and implementation are aimed at complex, multinational information sharing. Future development efforts should determine whether to incorporate a CDS design or use enterprise CDS services. Since it is very difficult to introduce CDS into a system during the maintenance phase, the CDS options have been moved to beyond level 3. See *NESI Part 2: Net-Centric ASD (NII) Checklist Guidance* for instructions on new development.

Current projects may be able to make use of a trusted CDS data service during maintenance upgrades, depending on the architecture. This could provide users with a single common data source, and could enable other applications running at different security levels to use the data.

For projects that are looking at major restructuring, use CDS enterprise services as they are developed and deployed.

4.2 Assessing scope

The Scope Assessment Matrix, below, correlates maintenance actions with net-centric upgrade actions. The matrix provides program managers with a rule-of-thumb guide for selecting net-centric upgrades based on the resources available for maintenance actions.

The matrix groups maintenance actions by resources available. It groups net-centric upgrades into the levels discussed above, based on risk and resource costs. For example, applications that only have the resources to perform fixes to Software Trouble Reports (STRs) would generally focus on Level 1 net-centric upgrades, achieving minimal net-centricity.

Table 2 – Scope Assessment Matrix

Maintenance actions	Resources available	Level 1, Minimum upgrade	Level 2, Mid-level upgrade	Level 3, Net-centric upgrade
Software Trouble Reports (STRs) only	$	X		
New functionality and STRs	$$	X	X	X (partial)
Application restructuring; major upgrades	$$$	X	X	X

These development effort levels do not address the DoD PKI Policy requirements.[6] Individual programs are responsible for that compliance.

Early net-centric upgrades will probably implement upgrade options from multiple levels, depending on the degree of net-centricity, robustness, and maturity of an application. In general, though, applications will start with a basic set of upgrades and evolve toward a full-featured, net-centric environment.[7]

Once STRs and Change Packages (CP) are prioritized from Configuration Control Board (CCB) reviews, use the table above and the maintenance options listed in the levels below to determine the effect of the maintenance actions proposed for your upgrade cycle. Once determined, use the matrix in section 6, *Mapping maintenance actions to enterprise technology objectives*, to assess the net-centric enabling accomplishments for this upgrade cycle.

[6] See Department of Defense Instruction 8520.2, Public Key Infrastructure (PKI) and Public Key (PK) Enabling, 1 April 2004.

[7] It is not necessary to complete all upgrades in one level before developing against subsequent levels. The state of an application will determine which upgrades can be done, in what order, and from which category.

4.3 Level 1: Minimum upgrade

This level contains the lower-cost, basic upgrades that enable the application to participate in a net-centric environment. The theme for Level 1 is to prepare the application for migration to the net-centric environment. It begins by factoring the application into tiers and insulating it from the enterprise. As part of this preparation, Level 1 recommends a number of self-assessments for specific technical issues. In addition, you can perform various configuration and provisioning changes that make the application easier to deploy and support across the enterprise.

The diagram below illustrates the options available for Level 1 upgrades.

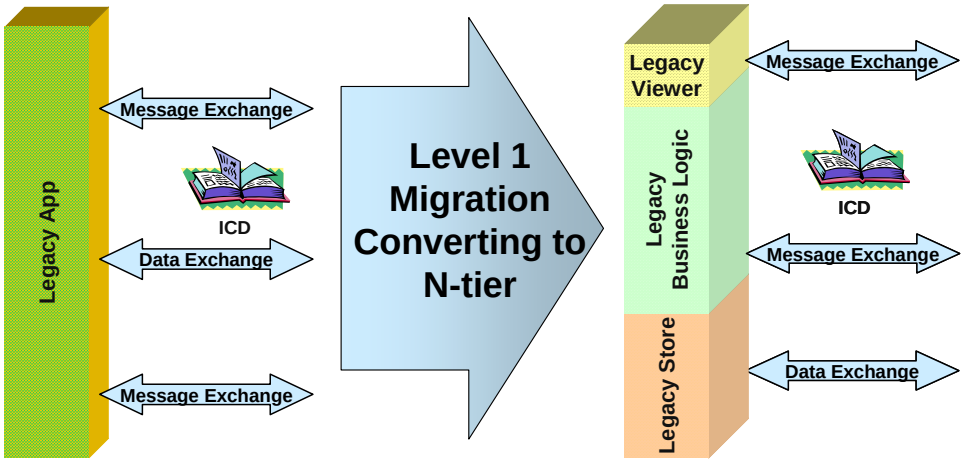


Figure 2 – Level 1 Migration: Conversion to N-Tier

4.3.1 Overarching

1. Identify additional development efforts for your specific environment, such as IT-21, GCCS, NMCI, and so on.
2. Develop and publish JUNIT[8] or automated tests, depending on the implementation language, for all public APIs.[9]
3. Assess the level of effort required to refactor the code into at least four tiers: client tier, presentation tier, middle tier, and data tier.
4. Migrate any operating-system-specific support to an abstraction layer and/or use POSIX[10]-compliant operating system (OS) APIs and test on currently supported operating systems and versions.
5. Develop independent version sequences for the application and the public application interfaces so that they can vary separately.
6. Create a configuration file helper class, the mechanisms to interact with the configuration file, and the configuration file. (This would be deployment descriptors for web-based

[8] http://www.junit.org

[9] Automated test drivers are available for various languages.

[10] Portable Operating System Interface.

applications or J2EE[11] applications, or a name/value-pair plain text configuration file for other languages. Microsoft applications should migrate off the registry.) You do not move all parameters to the configuration file in this iteration, though you should do so where it is reasonable. This is more of a placeholder mechanism to support subsequent levels.

7. Assess the level of effort required to support enterprise system management. Initially, applications must be able to send state information periodically or on demand, and receive commands. Communication occurs via an asynchronous communication mechanism.[12] Produce an enterprise management strategy document.[13]
8. Assess the application for the level of effort required to support availability. Produce an availability strategy document.
9. Assess the application or program for security. Produce a security policy document.
10. Incorporate a strong password scheme. Passwords should be at least eight characters long and contain at least one uppercase character, one numerical character, and one special character.

4.3.2 Client and presentation tier

1. Factor the GUI code into separable code that can be migrated to client and presentation tiers.
2. Publish and use public APIs.
3. Prepare existing APIs to migrate to separate tiers: client, presentation, middle, and data. Refactor existing APIs rather than writing new ones.
4. Decouple the public API from the rest of the application. Use a construct similar to Interfaces in Java, Abstract, Protocol classes in C++, or a design pattern[14] such as façade, proxy, adapter, or bridge.
5. Comment the API with Javadoc or a tool that produces Javadoc-type output.[15]
6. Migrate any code that accesses a collaborating system in the client tier to either the middle tier or the data tier. Wrap that code in a connector construct to isolate the application from the enterprise.
7. Assess the level of effort required to support portals for applications migrating to the Navy Enterprise Portal (NEP) or Air Force Portal. Use this to plan development efforts in a subsequent upgrade level.
8. Migrate client-side security features to middle and data tiers.
9. In Motif/X-window applications, implement a design pattern like façade, bridge, or proxy. This decouples the Motif from the rest of the application so that it supports service plug-ins.
10. Move obvious configuration parameters from this tier to the configuration file.

---

[11] Java 2 Platform, Enterprise Edition.

[12] For Navy programs, contact SSC San Diego, code 24202 Office for more information.

[13] For Navy programs, provide the document to the PEO C4I & Space Technical Director.

[14] *Design Patterns: Elements of Reusable Object-Oriented Software*, Gamma, Helm, Johnson, Vlissides, 1995, Addison-Wesley.

[15] Javadoc-type tools for other languages are available on the Internet.

11. Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters. Remove hard-coded IP addresses and URLs.
12. Add pre-condition checks to all public API parameters.[16]
13. Develop a plan for Discretionary Access Control (DAC) for each web-accessible component and for the migration to net-centric access controls such as RBAC.
14. Migrate all "magic number" constant values to named constants. In the C++ example below, the number 6 is a magic number.

```
//original code
if(numFiles > 6) {
// do something
}
//reworked code
const int MAXFILES = 6;
. . .
if(numFiles > MAXFILES) {
// do something
}
```

4.3.3 Middle tier

1. Publish and use the public APIs of the middle tier. The client tier uses this API.
2. Comment the API with Javadoc or a tool that produces Javadoc-type output.
3. Decouple the public API from the rest of the application, using a construct similar to Interfaces in Java or Abstract or Protocol classes in C++. This enables the implementation interface to vary independently from the part of the interface that is visible to the client.
4. Migrate any code that accesses a collaborating system in the middle tier to a connector construct. This isolates the application from the enterprise.
5. Develop connectors to the Directory NCES service through an appropriate design pattern. Implement the application side and incorporate a "Not Implemented" exception on the enterprise side.
6. Use SSL encryption to pass authentication information.
7. Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters.
8. Migrate all "magic number" constants to named constants.
9. Add pre-condition checks in all public APIs.
10. Move configuration parameters from this tier to the configuration file.
11. Migrate client-side security features to the middle tier.

[16] An example is the use of Assert() in C++.

4.3.4 Data tier

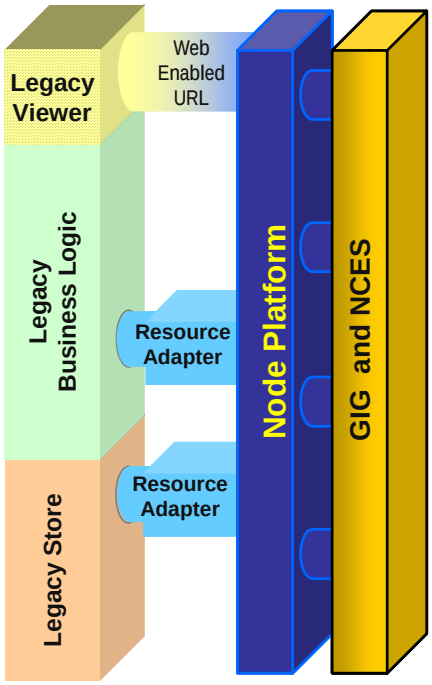
1. Publish and use public APIs. The middle tier uses this API.
2. Comment the API with Javadoc or a tool that produces Javadoc-type output.
3. Migrate any code that accesses a collaborating system in the data tier to a wrapper or connector construct. This isolates the application from the enterprise.
4. Migrate client-side security features to the data tier.
5. Develop connectors to the Directory NCES service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.
6. Use SSL encryption to pass authentication information.
7. Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters.
8. Migrate all “magic number” constants to named constants.
9. Add pre-condition checks in all public APIs.
10. Move configuration parameters from this tier to the configuration file.

4.4 Level 2: Mid-level upgrade

This level contains medium-cost upgrades that enable the application to participate in a net-centric environment at a higher level than Level 1 changes. The theme of Level 2 is to connect factored applications to the Node Platform Infrastructure.[17] The adapters connect the public APIs prepared in Level 1 to the Node Platform interfaces.

The diagram below illustrates the options available for Level 2 upgrades.

[17] See *NESI Part 4: Net-Centric Node Design Guidance*



Builds on Level 1 Capabilities

- **Do not change legacy viewer, application, or database**
- **Develop node interface to legacy application**
- **Expose data to network via interface to legacy store**
- **Allow users to task legacy application**
- **Allow users to receive status information from legacy application**

Figure 3 – Level 2 Migration: Access to Legacy Data and Applications

4.4.1 Overarching

1. Create a commercial "Install Anywhere" or "Install Shield" installation script. The script must be executable in multiple runtime environments. If your application resides on a target platform that this class of tools does not currently support, you can use another approach in the short term. In those cases, devise a migration strategy to open-source installation tools.
2. Incorporate XML-supporting infrastructure and administration.[18]
3. Collaborate with XML Namespace Managers to develop an XML representation for your application and COI data. Register this information in the DoD Metadata Repository and Clearinghouse.[19] This may affect the package names for class libraries and naming conventions.
4. Configure the application using an XML-type deployment descriptor model.[20]
5. If you are using an XML parser, develop a wrapper class around it. Code the application to that API to decouple the application from the XML parser.
6. Use validating XML parsers that support the XML schema standard.
7. Develop and publish automated systems integration tests for the entire application.
8. Provide backwards compatibility. Older clients should be able to exchange messages with newer services to execute older functionality.

---

[18] *Federal XML Developer's Guide*, http://xml.gov/documents/in_progress/developersguide.pdf.

[19] *DoD Metadata Registry and Clearinghouse*, http://xml.dod.mil.

[20] See the J2EE blueprints for more detail on implementation.

9. Provide forward compatibility. Newer clients should be able to exchange messages with older services to execute older functionality.
10. Develop a plan for porting and testing in target operational environments. Include the upgrades necessary to communicate with supporting system applications.
11. Migrate all environment variables to parameterization variables and store them in property files, deployment descriptors, or initialization files. For example, J2EE and web applications use deployment descriptors.
12. Finish migrating configuration parameters to the external configuration file.
13. Identify proprietary GOTS[21] and COTS[22] code and decouple it via wrapper classes. Design the wrapper classes.
14. Analyze functional areas of the application that will interface to the enterprise.[23]
15. Modify application structure to isolate change between the client tier and middle tier, per Level 1 assessment. Enable developers to modify and enhance discrete portions of the enterprise without affecting the others.
16. Develop connectors to the enterprise management service. Implement the application side and incorporate a "Not Implemented" exception on the enterprise side. At this level, incorporate self-diagnostics, enterprise management reports, and on/off functionality.[24]

4.4.2 Client and presentation tier

1. Migrate any programmatic security implementations to a container-managed security model. Do not use basic authentication for web-based applications.
2. Transform Windows-based applications to be Windows Logo-compliant.
3. Implement COE decoupling components specific to this tier.[25]
4. Develop portal support for applications migrating to the Navy Enterprise Portal (NEP) or Air Force Portal, based on earlier assessments. Base migration to the GIG Portal on JSR 168 for Java-based portlets.
5. For Motif/X-window applications, implement a design pattern like façade, bridge, or proxy. This decouples the Motif from the rest of the application in order to support service plug-ins per Level 1 assessments.

4.4.3 Middle tier

1. Develop connectors to the Messaging enterprise service. Implement the application side and incorporate a "Not Implemented" exception on the enterprise side.

---

[21] Government off-the-shelf.

[22] Commercial off-the-shelf.

[23] Example include user management and authentication.

[24] Consider Java-based JMX technologies. See http://java.sun.com/products/JavaManagement.

[25] The high-level COE migration strategy is outlined in section 5, *Migrating COE systems and applications*.

2. Develop connectors to the IA/Security enterprise service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.
3. Develop connectors to the Discovery enterprise service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.
4. Develop connectors to the Net Time enterprise service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.
5. Develop connectors to the Network Management enterprise service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.
6. Develop connectors to an external directory service for authentication. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side. Some application servers have a realm database as part of the application server. This realm database must be “pluggable” to support the use of other directory servers.
7. Migrate any programmatic security implementations to a container-managed security model. Do not use basic authentication for web-based applications.
8. For Java applications, develop Discretionary Access Control based on container-managed security and the enterprise connector frameworks. Isolate the access control for migration to net-centric RBAC authorization services.
9. For C, C++, and ADA applications, develop Discretionary Access Control based on a container-managed security model using LDAP. Isolate the access control for migration to net-centric RBAC authorization services.
10. Migrate from raw sockets and primitive connection APIs to an abstraction layer.
11. Implement container-managed transactions and a concurrency control model.
12. Implement COE decoupling components specific to this tier.
13. Implement application integration and backend integration with initial collaborators.

4.4.4 Data tier

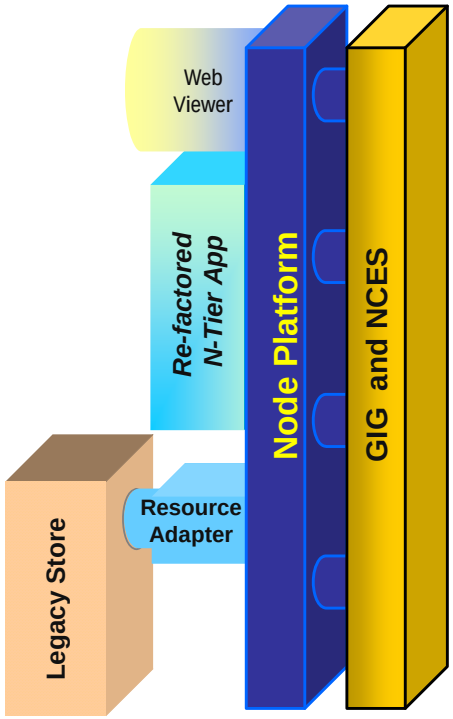
1. Collaborate with Data Area Managers to develop a strategy to incorporate enterprise data policies. Identify, catalog, and report Data Area Manager requirements, including: data formats; database and versions; authoritative data sources; stored procedures and triggers; and data latency and integrity issues.
2. Coordinate shared resources with collaborators.
3. Implement COE decoupling components specific to this tier.
4. Implement container-managed transactions and a concurrency control model.
5. Create backwardly compatible data mappings for messages.
6. Remove global accounts from databases and integrate them into authentication/access control components.
7. Implement a data integrity scheme for ensuring correct data management when the database is accessed from multiple locations.
8. Implement application integration and backend integration with initial collaborators.

9. Create and publish content metadata in accordance with the guidance from the DoD Metadata Registry and Clearinghouse.[26]
10. Isolate the application's data tier from the rest of the application with an open-standards CLI interface layer like ODBC, JDBC, a RogueWave-like layer, or an equivalent abstraction.
11. For database applications, migrate from proprietary SQL to ANSI STD SQL 92 or ANSI STD 99, depending on the database. If the application may lose functionality by migrating off proprietary SQL, you may use an alternative approach. For data stream applications, develop a wrapper abstraction layer that insulates proprietary protocols from the rest of the application.
12. Define the multinational sharing requirements for the data that the service will create and use.

4.5 Level 3: Net-centric upgrade

This level contains the higher-cost upgrades that enable the application to participate fully in a net-centric environment. The theme for Level 3 is to migrate the refactored application code to the Node Platform Infrastructure and prepare the application for the net-centric enterprise environment (e.g., namespaces, XML, metadata, publish-subscribe interfaces).

The diagram below illustrates the options available for Level 3 upgrades.



Builds on Level 2 Capabilities

- **Rewrite parts (or all) of legacy application as N-tier application (J2EE or .NET)**
- **Make application part of node workflow**
- **Access application information via URL or through web portal using a browser**
- **Allow publish and subscribe with web alerts**

Figure 4 – Level 3 Migration: Transform Legacy Applications

[26] *DoD Metadata Registry and Clearinghouse*, http://xml.dod.mil.

4.5.1 Overarching

1. Develop a remote administration/management model that includes frameworks and connectors for remote monitoring, server resource management, and remote software upgrades and maintenance.[27]
2. Develop remote policies for administrators, operators, and developers.
3. Develop remote installation procedures for components and applications.
4. Implement enterprise authentication and Single Sign-on, using the connectors developed earlier to facilitate access to data and logic. Use XML-based security assertions to pass authentication information.
5. Integrate application components with the enterprise namespace strategy.
6. Implement return-code-to-exception mapping for applications that use return codes.
7. Place exceptions in the public API descriptions.
8. Develop non-real-time to real-time bridge designs with collaborators.
9. Implement initial implementation of enterprise data policies.
10. Test the application on target operating systems in your current and planned operational environments. Include all supporting system applications.
11. Test the application on middleware, including application servers and ORBs, in your current and planned operational environments. Include all supporting system application interactions.
12. Develop and publish automated acceptance tests for the entire application.
13. Develop proprietary GOTS and COTS wrapper classes and integrate them into the application.
14. Modify the application structure to isolate change between the middle tier and data tier, per Level 1 assessments. Enable developers to modify and enhance discrete portions of the enterprise without affecting the others.
15. Implement consistent XML data formats, services such as WSDL, and protocols such as SOAP to support data and service exchange across distributed nodes.
16. Assess data for integration with CoI or enterprise language and ontologies (e.g., C2IEDM).
17. Implement the enterprise management connector back ends. Integrate with the enterprise management service for on/off, heartbeat, and reports.

4.5.2 Client and presentation tier

1. Migrate decoupled clients to web-page or decoupled-thick-client GUIs. Ensure you can download them independently from the application. The client module must be able to communicate with the presentation tier on the server via SSL.
2. Restructure the source code to use the enterprise namespace strategy.
3. Implement application integration and backend integration with more collaborators.

---

[27] Software must be able to be installed over the network.

4. Implement return-code-to-exception mapping.
5. Begin implementing enterprise data policies.
6. Develop and publish automated acceptance tests.
7. Develop the enterprise interface connectors and integrate them into the application.

4.5.3 Middle tier

1. Develop a non-repudiation scheme for application service-to-service interactions.
2. Implement a services-based access model for business logic and data, with support for legacy in/out messages. Exchange node-to-node information through services in the middle tier.
3. Migrate the business logic from the data tier and client tier into the middle tier. Some of the business logic may be contained in the database and not affect the risk strategy.
4. Implement return-code-to-exception mapping.
5. Integrate the application namespace with the enterprise namespace strategy.
6. Implement application integration and backend integration with more collaborators.
7. Begin implementing enterprise data policies.
8. Develop and publish automated acceptance tests.

4.5.4 Data tier

1. Migrate data tier items from the client and middle tiers into the data tier.
2. Implement return-code-to-exception mapping.
3. Implement a services-based data access model with support for legacy in/out messages.
4. Implement an application collaboration and mediation framework.
5. Incorporate XML-supporting infrastructure and administration.
6. Integrate the application namespace with the enterprise namespace strategy.
7. Implement application integration and backend integration with more collaborators.
8. Begin implementing enterprise data policies.
9. Develop and publish automated acceptance tests.
10. For raw byte-stream data applications and sensors, create an object-oriented wrapper abstraction layer.

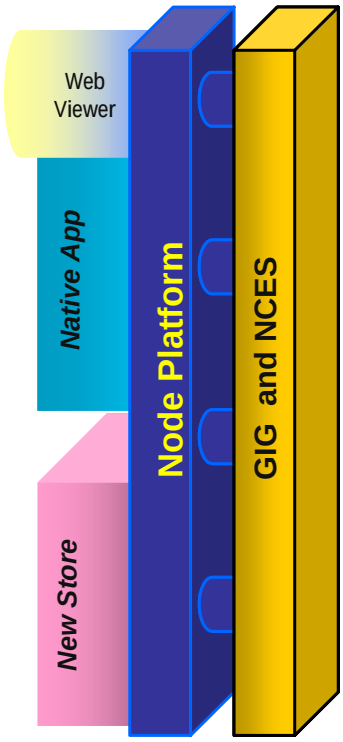
4.6 Level 4: Full net-centric integration

Migrating legacy applications to a net-centric environment after Level 3 will require major development efforts.[28] The theme of this level is to take a factored, tiered application and

[28] See *NESI Part 1: Net-Centric Overview* and *NESI Part 2: Net-centric ASD (NII) Checklist Guidance*

provide implementation-independent (e.g., XML-based) information exchange, new services for other nodes, and the ability to consume services provided by other nodes.

The diagram below illustrates the options available for upgrades beyond Level 3.



Builds on Level 3

- **Use web technologies to create a service based architecture**
- **Replace legacy viewers, applications and databases with homogeneous SOA and data strategy**
- **Develop new applications to run in homogeneous node environment.**
- **Access applications via the web portal and the web services**
- **Interface to full enterprise workflow and collaboration environment**
- **Make web services available to external systems for machine-to-machine data and application exchanges**

Figure 5 – Level 4: Full Net-Centric Integration

4.6.1 Overarching

1. For applications using RDBMS replication, develop an RDBMS replication strategy to migrate to third-party replication providers and decouple from proprietary replication engines. The replication technology must support replication across vendor databases and versions.
2. Develop an application lifecycle framework that shows how the application interacts with the enterprise.
3. Implement or interface to enterprise caching and communications servers.
4. Implement a cross-domain solution or use CDS enterprise services.
5. Develop a strategy to emulate or gateway legacy networks so that all users are perceived as IP nodes on a larger network.
6. Implement a non-real-time to real-time bridge with collaborators.
7. Implement XML-supporting infrastructure and administration for the enterprise.
8. Implement remote installation of components and applications.
9. Implement availability- and fault-tolerant services.
10. Implement load balancing.

11. Implement database integration.
12. Implement connectors to information management components. These should include:
    - Indexing – content metadata
    - Searching – to parse indexed material
    - Preferences/customization/personalization – what you see in your templates
    - Profiling
13. Develop interfaces and connectors to Content Delivery Network collaborators.
14. Develop interfaces and connectors to Intelligent Agents.
15. Develop interfaces and connectors to a Profile Management system.
16. Develop interfaces and connectors to a Workflow Management system.
17. Develop interfaces and connectors to a Process Management system.
18. Develop interfaces and connectors to Local Management subsystems.
19. Support strong authentication techniques using centralized authentication servers.

5 Migrating COE systems and applications

As the DoD moves toward net-centricity, systems and applications may migrate away from the Common Operating Environment (COE). Migrating COE systems to a net-centric infrastructure requires analyzing the system's dependencies on the COE. Some systems are built to run on the COE without major dependencies; others use complex COE-based functionality. The guidance NESI provides on developing net-centric systems will help bridge the gap.

5.1 Selecting an approach

COE applications will migrate in phases and at various levels of decoupling. The spectrum of legacy integration and transition possibilities requires multiple integration approaches. A customized approach should mitigate the level of risk and leverage maintenance dollars.

The best approach for a given system depends on two major factors:

- Will the system have different users, requirements, and interfaces than it does now? If so, you may need more flexibility and resources for the transition.
- What level of COE integration does the system have? How is the implementation achieved? What COE components does the system use? Are there plans for developing non-COE versions of those components?

COE components include workstation and user interface facilities, message processing facilities, and mechanisms for software builds and configuration management.

As the net-centric environment expands, COE functionality will be iteratively replaced by services. Over time, fewer and fewer applications will rely on the COE.

5.2 Analyzing COE capabilities

The COE provides a number of basic, DISA-provided capabilities, listed below. Use these as the starting point for mapping and transitioning COE capabilities to net-centric, open-standards capabilities.

- An operating system environment
- A GOTS COE kernel that includes:
    - A packaging and installation technology; COE segmentation, and the COE Installer
    - Security templates for Discretionary Access Control
    - Account Manager
    - Profile Manager
    - Auditing
    - Process management
    - Platform configuration management

The COE also offers a number of COTS and GOTS products for applications. The migration strategy should consider the number and level of dependencies on GOTS components that are not available outside the COE environment. The major COE GOTS components of interest are:

- Alerts Services
- Joint Mapping Tool Kit (JMTK)
- Common Message Processor (CMP)
- Integrated C4I System Framework (ICSF)

Each of these products requires a particular migration strategy.

5.3 Decision tree

Use the following decision tree to determine the best migration option for each COE capability.

1. If your system currently supports multiple environments, and you choose to continue multi-OS (heterogeneous) support, you should provide some form of common interface similar to the COE interface. You can:
    - Build your own version of that capability.
    - Port forward the existing COE implementation.
    - Buy a commercial product to accomplish the same function and migrate your system to that product.
2. If your system currently supports multiple environments, and you choose to support only a single environment, you can:
    - Buy and use the commercial solution set for that environment (e.g., the Navy's NMCI approach).
    - Build an abstraction layer above that environment to isolate your system. Towards this end, you can either:
        - Build your own version of that capability.
        - Port forward the existing COE implementation.
        - Buy a commercial product to accomplish the same function and migrate your system to that product.
3. If your system currently supports multiple environments, and you choose to implement multiple solutions, one per environment, these solutions will not be interoperable. For each environment, you can:
    - Buy and use the commercial solution set for that environment.
    - Build an abstraction layer above that environment to isolate your system. You can either:
        - Build your own version of that capability.
        - Port forward the existing COE implementation.

- Buy a commercial product to accomplish the same function and migrate your system to that product.

5.4 Examples of mixed COE/non-COE systems

For insight into COE migration techniques, examine systems that have developed or are developing mixed COE/non-COE systems:

- I3 integration framework developers toolkit[29]
- Joint Enterprise DoDIIS Infrastructure (JEDI)[30]
- Extensible Tactical C4I Framework (XTCF)[31]

5.5 Migrating systems with basic COE dependency

Systems with a basic COE dependency only rely on the COE installer and the GOTS COE kernel. The migration strategy in this case is relatively straightforward.

1. Identify the standard COTS components used (e.g., RDBMS) and provide for non-COE versions. These may be provided by your system or by the node on which your system runs.
2. Remove any COE segmentation and COE installer components. Use tools such as MakeInstall and UnMakeInstall to make your system a segment or not a segment, as appropriate.
3. For GOTS, develop an installation procedure using commercial installation technology (e.g., InstallShield for Windows, InstallAnywhere for multiple platforms).
4. Implement Logo compliance on Windows.
5. Implement Appcert compliance on Solaris.
6. Reserve and deconflict machine resources, file system conventions, environment variables, and port numbers.
7. Use only published APIs for the OS.
8. Provide for user and group account management. Set file and directory permissions and password management.
9. Provide process management configuration rules such as what processes run and when. Note that COE-specific process management differs from what the target environment provides.
10. After migration, if your system will provide its own OS and hardware, use the NSA-developed COE security lockdown directions to develop an equivalent security lockdown procedure as part of installation. The general approach is to lock down everything and document those functions that need to be unlocked.

---

[29] Contact the GCCS-M Program Management Office.

[30] Contact the JEDI Program Management Office, AFRL/IFEB, http://extranet.if.afrl.af.mil/jedi/ or mailto:jedi@rl.af.mil.

[31] Contact the XTCF Program Management Office.

11. After migration, if your system will be hosted on another system, develop the security configuration required for proper operation in that environment (e.g., which ports must be open).

5.6 Migrating systems with COE component dependencies

Review the COE components in this section and identify which ones your system depends on. Develop a specific migration strategy for each one. Each section below makes suggestions that you can use as a starting point. Your specific requirements determine the most cost-effective approach.

Some components require multiple migration steps. Others require interim development tactics such as COE to non-COE bridge segments. There may be alternatives to the suggested strategies for some components.

5.6.1 COE alerts dependency

- Investigate alternate NCES, OS, or node alert/notification/messaging processing.

5.6.2 COE APM/CDS dependency

- Provide APM/CDS as a plug-in (subset appropriate to application usage).
- Replace individual invocations with alternate NCES, OS, or node service calls.

5.6.3 COE JMTK[32] dependency

- Port any needed JMTK to a non-COE environment. There are several ongoing activities to get off the JMTK.
- Investigate porting your system to use C/JMTK[33] or alternate mapping packages such as an OGC-visualization-independent layer.

5.6.4 COE CMP dependency

- Investigate IRIS.[34]
- Investigate a CMP migration strategy.

5.6.5 COE ICSF dependency

- Include the TMS (CST, etc.), UCP (netproc, etc.), and JMTK subcomponents. This is typically a complex effort.
- Analyze any other ICSF-specific applications that are used.
- Investigate ongoing web-enabling efforts such as WebCOP.
- Investigate the DISA-sponsored User Defined Operation Picture (UDOP) program.

---

[32] See http://www.jmtk.org for details on JMTK.

[33] See http://www.cjmtk.com for details on C/JMTK.

[34] See http://www.sseusa.com for details on IRIS.

5.7 Migrating COE components using a bridge approach

For many COE components, a bridge design pattern[35] approach may be appropriate.

In the bridge approach, COE capabilities continue to exist. A new companion segment, called a bridge segment, is installed on the COE system. This segment provides an interface by which net-centric systems can access the COE-based functionality. It also provides net-centric services that make the component functionality available on the network. New, non-COE applications invoke the bridge segment services to access the underlying COE-based functionality.

Depending on the number of public APIs that have to be rewritten, this interim strategy may be more cost-effective than strategies such as wholesale segment conversion. The following diagrams illustrate the bridge approach.

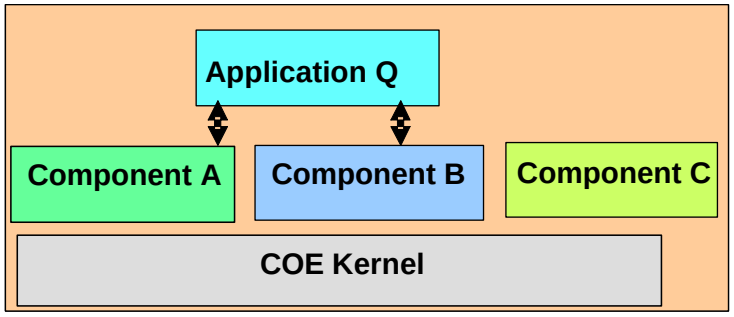


Figure 6 – Notional COE-Based System

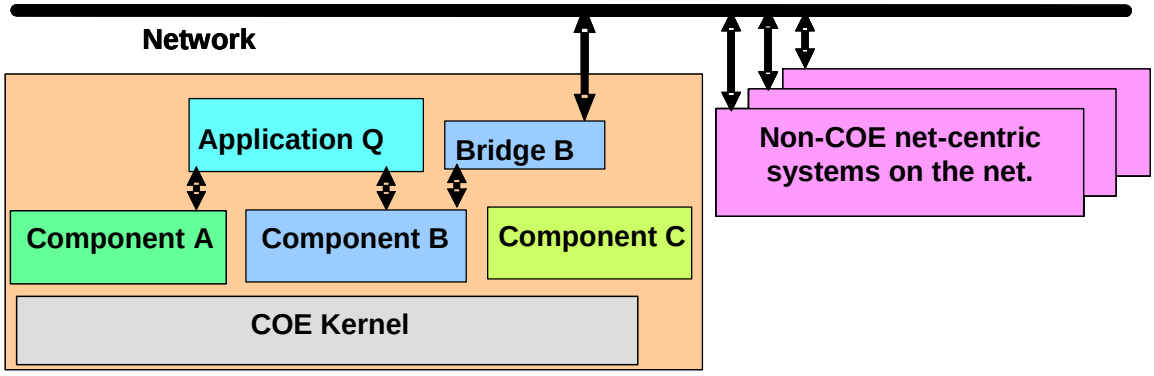


Figure 7 – Notional COE Component Hybrid Bridge Configuration

[35] *Design Patterns: Elements of Reusable Object-Oriented Software*, Gamma, Helm, Johnson, Vlissides, 1995, Addison-Wesley.

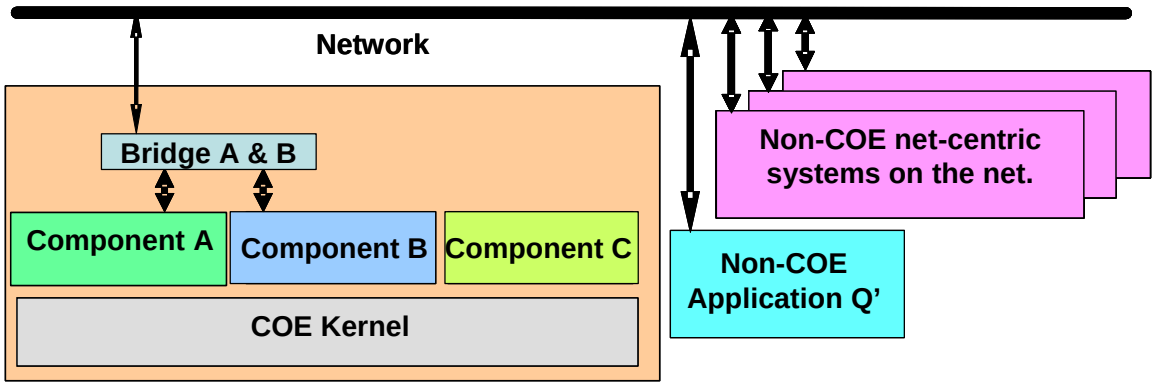


Figure 8 – Notional COE Application Hybrid Bridge Configuration

6 Mapping maintenance actions to enterprise technology objectives

This section maps the maintenance actions identified above to each of the enterprise technology objectives listed in the *NESI Part 1: Net-Centric Overview*. The table illustrates how the levels of integration shown above flow into each other and support the flow among maintenance actions.

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
Level 1 Overarching									
1	Identify additional development efforts for your specific environment, such as IT-21, GCCS, NMCI, and so on.	X	X	X	X	X	X	X	X
2	Develop and publish JUNIT or automated tests, depending on the implementation language, for all public APIs.	X	X	X				X	
3	Assess the level of effort required to refactor the code into at least four tiers: client tier, presentation tier, middle tier, and data tier.	X		X				X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
4	Migrate any operating-system-specific support to an abstraction layer and/or use POSIX-compliant OS APIs and test on currently supported operating systems and versions.	X	X	X				X	
5	Develop independent versioning for the application and the public application interfaces so that they can vary separately.	X	X	X				X	
6	Create a configuration file helper class, the mechanisms to interact with the configuration file, and the configuration file.	X		X		X		X	
7	Assess the level of effort required to support enterprise system management.	X	X			X	X		X
8	Assess the application for the level of effort required to support availability.	X	X		X	X	X		X
9	Assess the application or program for security. Produce a security policy document.	X	X		X		X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
10	Incorporate a strong password scheme.	X	X		X		X		X
Level 1 Client/Presentation Tier									
1	Factor the GUI code into separable code that can be migrated to client and presentation tiers	X	X					X	
2	Publish and use the public APIs	X	X	X		X		X	X
3	Prepare existing APIs to migrate to separate tiers: client, presentation, middle, and data.	X	X	X		X		X	X
4	Decouple the public API from the rest of the application. Use a construct similar to Interfaces in Java, Abstract or Protocol classes in C++, or a design pattern such as façade, proxy, adapter, or bridge.	X	X	X				X	
5	Comment the API with Javadoc or a tool that produces Javadoc-type output.	X	X	X		X		X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
6	Migrate any code that accesses a collaborating system in the client tier to either the middle tier or the data tier. Wrap that code in a connector construct to isolate the application from the enterprise.	X	X	X				X	
7	Assess the level of effort required to support portals for applications migrating to the Navy Enterprise Portal (NEP) or Air Force Portal. Use this to plan development efforts in a subsequent upgrade level.	X		X	X				
8	Migrate client-side security features to middle and data tiers.	X	X		X		X		X
9	In Motif/X-window applications, implement a design pattern like façade, bridge, or proxy. This decouples the Motif from the rest of the application so that it supports service plug-ins.	X	X	X				X	
10	Move obvious configuration parameters from this tier to the configuration file.	X		X		X		X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
11	Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters. Remove hard-coded IP addresses and URLs.	X		X		X		X	
12	Add pre-condition checks to all public API parameters.	X	X	X			X	X	X
13	Develop a plan for Discretionary Access Control (DAC) for each web-accessible component and for the migration to net-centric access controls such as RBAC.	X	X		X		X		X
14	Migrate all “magic number” constant values to constant variables.			X		X		X	
Level 1 Middle tier									
1	Publish and use the public APIs of the middle tier.	X	X	X				X	
2	Comment the API with Javadoc or a tool that produces Javadoc-type output.	X	X	X				X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
3	Decouple the public API from the rest of the application, using a construct similar to Interfaces in Java or Abstract or Protocol classes in C++.	X	X	X				X	
4	Migrate any code that accesses a collaborating system in the middle tier to a connector construct.	X	X			X	X		X
5	Develop connectors to the Directory NCES service through an appropriate design pattern. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.	X	X	X				X	
6	Use SSL encryption to pass authentication information.	X	X		X		X		X
7	Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters.	X		X		X		X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
8	Migrate all “magic number” constants to constant variables.			X		X		X	
9	Add pre-condition checks in all public APIs.	X	X	X			X	X	X
10	Move configuration parameters from this tier to the configuration file.	X		X		X		X	
11	Migrate client-side security features to the middle tier.	X	X		X		X		X
Level 1 Data Tier									
1	Publish and use public APIs.	X	X	X				X	
2	Comment the API with Javadoc or a tool that produces Javadoc-type output.	X	X	X				X	
3	Migrate any code that accesses a collaborating system in the data tier to a wrapper or connector construct.	X	X			X			X
4	Migrate client-side security features to the data tier.	X	X		X		X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
5	Develop connectors to the Directory NCES service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.	X	X						X
6	Use SSL encryption to pass authentication information.	X	X		X		X		X
7	Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters.	X		X		X		X	
8	Migrate all “magic number” constants to constant variables.			X		X		X	
9	Add pre-condition checks in all public APIs.	X	X	X			X	X	X
10	Move configuration parameters from this tier to the configuration file.	X		X		X		X	
Level 2 Overarching									
1	Create a commercial “Install Anywhere” or “Install Shield” installation script.	X	X					X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
2	Incorporate XML-supporting infrastructure and administration.	X	X	X				X	
3	Collaborate with XML Namespace Managers to develop an XML representation for your application and COI data.	X	X	X				X	
4	Configure the application using an XML-type deployment descriptor model.	X	X	X				X	
5	If you are using an XML parser, develop a wrapper class around it. Code the application to that API to decouple the application from the XML parser.	X	X	X				X	
6	Use validating XML parsers that support the XML schema standard.	X	X	X			X		X
7	Develop and publish automated systems integration tests for the entire application.	X	X	X				X	
8	Provide backwards compatibility.	X	X			X	X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
9	Provide forward compatibility.	X	X			X	X		X
10	Develop a plan for porting and testing in target operational environments. Include the upgrades necessary to communicate with supporting system applications.	X	X	X				X	
11	Migrate all environment variables to parameterization variables and store them in property files, deployment descriptors, or initialization files.	X	X	X				X	
12	Finish migrating configuration parameters to the external configuration file.	X	X	X				X	
13	Identify proprietary GOTS and COTS code and decouple it via wrapper classes. Design the wrapper classes.	X	X	X				X	
14	Analyze functional areas of the application that will interface to the enterprise.	X	X				X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
15	Modify application structure to isolate change between the client tier and middle tier, per Level 1 assessment.	X	X	X				X	
16	Develop connectors to the enterprise management service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side. At this level, incorporate self-diagnostics, enterprise management reports, and on/off functionality.	X	X			X	X		X
Level 2 Client and Presentation Tiers									
1	Migrate any programmatic security implementations to a container-managed security model. Do not use basic authentication for web-based applications.	X	X		X	X	X		
2	Transform Windows-based applications to be Windows Logo-compliant.	X	X	X				X	
3	Implement COE decoupling components specific to this tier.	X	X	X			X	X	X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
4	Develop portal support for applications migrating to the Navy Enterprise Portal (NEP) or Air Force Portal, based on earlier assessments. Base migration to the GIG Portal on JSR 168 for Java-based portlets.		X		X	X			
5	For Motif/X-window applications, implement a design pattern like façade, bridge, or proxy.	X	X	X				X	
Level 2 Middle tier									
1	Develop connectors to the Messaging NCES service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.	X	X		X	X	X	X	X
2	Develop connectors to the Audit NCES Services.	X	X	X	X	X	X	X	X
3	Develop connectors to the Discovery NCES service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.	X	X	X			X	X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
4	Develop connectors to the Net Time enterprise services. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.	X	X				X		X
5	Develop connectors to an external directory service for authentication. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side. Some application servers have a realm database as part of the application server. This realm database must be “pluggable” to support the use of other directory servers.	X	X		X		X		X
6	Migrate any programmatic security implementations to a container-managed security model. Do not use basic authentication for web-based applications.	X	X	X		X		X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
7	For Java applications, develop Discretionary Access Control based on container-managed security and the enterprise connector frameworks. Isolate the access control for migration to net-centric RBAC authorization services.	X	X	X	X		X		X
8	For C, C++, and ADA applications, develop Discretionary Access Control based on a container-managed security model using LDAP. Isolate the access control for migration to net-centric RBAC authorization services.	X	X	X	X		X		X
9	Migrate from raw sockets and primitive connection APIs to an abstraction layer.	X	X	X				X	
10	Implement container-managed transactions and a concurrency control model.	X	X		X		X		X
11	Implement COE decoupling components specific to this tier.	X	X	X			X	X	X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
12	Implement application integration and backend integration with initial collaborators.	X	X	X					X
Level 2 Data Tier									
1	Collaborate with Data Area Managers to develop a strategy to incorporate enterprise data policies.	X	X				X		X
2	Coordinate shared resources with collaborators.	X	X			X	X		X
3	Implement COE decoupling components specific to this tier.	X	X	X			X	X	X
4	Implement container-managed transactions and a concurrency control model.	X	X	X	X	X	X		X
5	Create backwardly compatible data mappings for messages.	X	X	X		X	X		X
6	Remove global accounts from databases and integrate them into authentication/ access control components.	X	X		X	X	X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
7	Implement a data integrity scheme for ensuring correct data management when the database is accessed from multiple locations.	X	X		X	X	X		X
8	Implement application integration and backend integration with initial collaborators.	X	X	X	X		X		X
9	Create and publish content metadata in accordance with the guidance from the *DoD Metadata Registry and Clearinghouse*.	X	X	X				X	X
10	Isolate the application's data tier from the rest of the application with an open-standards CLI interface layer like ODBC, JDBC, a RogueWave-like layer, or an equivalent abstraction.	X	X	X				X	
11	For database applications, migrate from proprietary SQL to ANSI STD SQL 92 or ANSI STD 99, depending on the database.	X	X	X				X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
12	Define the multinational sharing requirements for the data that the service will create and use.	X	X						
Level 3 Overarching									
1	Develop a remote administration/management model that includes frameworks and connectors for remote monitoring, server resource management, and remote software upgrades and maintenance.	X		X				X	
2	Develop remote policies for administrators, operators, and developers.	X	X		X		X	X	X
3	Develop remote installation procedures for components and applications.	X	X	X				X	
4	Implement enterprise authentication and Single Sign-on, using the connectors developed earlier to facilitate access to data and logic. Use XML-based security assertions to pass authentication information.	X	X		X	X	X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
5	Integrate application components with the enterprise namespace strategy.	X	X	X		X			X
6	Implement return-code-to-exception mapping for applications that use return codes.	X	X		X	X	X		X
7	Place exceptions in the public API descriptions.			X				X	
8	Develop non-real-time to real-time bridge designs with collaborators.	X	X				X		X
9	Implement initial implementation of enterprise data policies.	X	X			X	X		X
10	Test the application on target operating systems in your current and planned operational environments. Include all supporting system applications.	X	X	X					

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
11	Test the application on middleware, including application servers and ORBs, in your current and planned operational environments. Include all supporting system application interactions.	X	X	X					
12	Develop and publish automated acceptance tests for the entire application.	X	X	X					
13	Develop proprietary GOTS and COTS wrapper classes and integrate them into the application.	X	X	X				X	
14	Modify the application structure to isolate change between the middle tier and data tier, per Level 1 assessments.	X	X	X				X	
15	Implement consistent XML data formats, services such as WSDL, and protocols such as SOAP to support data and service exchange across distributed nodes.	X	X			X	X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
16	Assess data for integration with CoI or enterprise language and ontologies (e.g., C2IEDM).	X	X			X	X		X
17	Implement the enterprise management connector back ends. Integrate with the enterprise management service for on/off, heartbeat, and reports.	X	X			X	X		X
Level 3 Client and Presentation Tiers									
1	Migrate decoupled clients to web-page or decoupled-thick-client GUIs. Ensure you can download them independently from the application. The client module must be able to communicate with the presentation tier on the server via SSL.	X	X					X	
2	Restructure the source code to use the enterprise namespace strategy.	X	X	X				X	
3	Implement application integration and backend integration with more collaborators.	X	X			X		X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
4	Implement return-code-to-exception mapping.	X	X		X	X	X		X
5	Begin implementing enterprise data policies.	X	X			X	X		X
6	Develop and publish automated acceptance tests.	X	X	X				X	
7	Develop the enterprise interface connectors and integrate them into the application.	X	X		X	X	X		X
Level 3 Middle tier									
1	Develop a non-repudiation scheme for application service-to-service interactions.	X	X				X		X
2	Implement a services-based access model for business logic and data, with support for legacy in/out messages. Exchange node-to-node information through services in the middle tier.	X	X			X	X		X
3	Migrate the business logic from the data tier and client tier into the middle tier.	X	X	X				X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
4	Implement return-code-to-exception mapping.	X	X		X	X	X		X
5	Integrate the application namespace with the enterprise namespace strategy.	X	X	X				X	
6	Implement application integration and backend integration with more collaborators.	X	X			X			X
7	Begin implementing enterprise data policies.	X	X			X	X		X
8	Develop and publish automated acceptance tests.	X	X	X				X	
Level 3 Data Tier									
1	Migrate data tier items from the client and middle tiers into the data tier.			X				X	
2	Implement return-code-to-exception mapping.	X	X		X	X	X		X
3	Implement a services-based data access model with support for legacy in/out messages.	X	X			X	X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
4	Implement an application collaboration and mediation framework.	X	X		X		X		X
5	Incorporate XML-supporting infrastructure and administration.	X	X	X	X	X	X	X	X
6	Integrate the application namespace with the enterprise namespace strategy.	X	X	X		X	X	X	X
7	Implement application integration and backend integration with more collaborators.	X	X			X	X		X
8	Begin implementing enterprise data policies.	X	X				X		X
9	Develop and publish automated acceptance tests.	X	X	X				X	
10	For raw byte-stream data applications and sensors, create an object-oriented wrapper abstraction layer.	X	X			X	X		X